WEB SERVICES-BASED QUERY REWRITING AND RESOLUTION IN LOOSELY COUPLED INFORMATION SYSTEMS

Mahmoud Barhamgi, Pierre-Antoine Champin and Djamal Benslimane
LIRIS Laboratory, Claude Bernard Lyon1 University, 69622 Villeurbanne, France

Keywords: Web services composition, Data integration, P2P systems.

Abstract: Recently a great deal of information resources has been exposed in the form of Data-Providing Web services in Peer-to-peer based e-collaboration environments. In this paper we model these services as RDF views over the peer local ontology, then we use these views in establishing a composition of Web services that satisfies a received query. Our composition is Data-Driven, therefore we apply extra data treatments on data flow between composed services before getting the desired result.

1 INTRODUCTION

In today's loosely-coupled e-collaboration environments (e.g. eHealth, eGov...etc) the access to an increasing number of data sources is made through Web services. We call this kind of services as "*Data-Providing Services*" as opposed to "*Functionality-Providing Services*" since their invocation only returns a piece of information without causing any change in the environment (e.g. charging a credit card, ...etc). Data-Providing services are very common in eHealth collaboration environments, in the same health site, for example, they are used to encapsulate and integrate numerous proprietary data sources that otherwise cannot be integrated, e.g. sensors, equipments equipped with proprietary interfaces...etc. Across different sites, they are used to share patients records, or as a means to transfer outsourced data. Such e-collaboration environments can be modeled as peer-to-peer environments where every peer holds a collection of DP services, puts them at the disposal of its partners and, in return, they provide it with their DP services. The collaboration implies that some of the peer's data items are outsourced or stored at its partners and that it needs to exploit their DP services to retrieve these items when needed.

So far, P2P Data Management and Integration Systems (Halevy et al., 2004; Rodríguez-Gianolli et al., 2005; Löser et al., 2003; Spyropoulou and Dalamagas, 2006) were only concerned with handling traditional data sources (as opposed to services). In these systems, peers are supposed to directly hold and expose their data, either in a syntactic form (XML), or recently in some semantic forms (OWL instances plus some inferencing capabilities), then when they are interrogated, they apply queries squarely to data instances in order to materialize answers. However, none of these systems pay attention to the consequences raised by the adoption of DP services for data sharing. With this form of data accessibility, it becomes impossible to materialize data in any forms or structures before applying, in a subsequent step, queries to it, rather the query resolution here necessitates to decompose the received query in terms of available services, compose these services, and to coordinate their execution.

2 OUR APPROACH

In this paper we provide a condensed description of our framework ((Barhamgi et al., 2007)) to provide better support for data sharing and integration in *e*Systems that make an extensive use of DP services. In our framework individual peers adopt the stack presented in figure 1 which has the following layers.
1. ***Data-Providing Services Layer***. This layer holds (or makes reference to) services that contribute data items pertaining to the peer in question. These services are either local or remote ones (i.e offered by the peer's partners), and they can be picked up based on the SOA(Papazoglou, 2003) model.
2. ***Views Layer***. In this layer we model previously

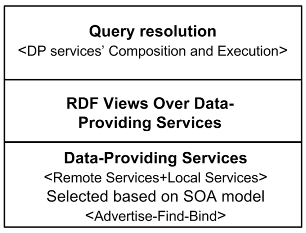

Query resolution
<DP services' Composition and Execution>
RDF Views Over Data-Providing Services
Data-Providing Services
<Remote Services+Local Services>
Selected based on SOA model
<Advertise-Find-Bind>

Figure 1: The Stack adopted by our peers for query resolution in *e*Systems adopting Data-Providing services.

selected services (in the first layer) as *RDF parameterized views* over the peer's local ontology. These views serve us for query resolution in the next layer.
3. ***Data-Providing Services Composition and Execution Layer***. Here we make use of previously defined views to decide what are the services whose composition can satisfy a received query (either a local query or a query received from the peer's acquaintances).

The rest of the paper is organized as follows. Next we provide a running example. In section 4 we model both our query and services. In section 5 we discuss the issue of query rewriting then we conclude the paper in section 6.

3 RUNNING EXAMPLE

Consider the case of the peer P1, it holds the ontology in figure 2, and it has the following DP services. Two remote DP services WS_1 and WS_2 to retrieve *Test A ,Test B* (specializations of Test) respectively. Two local DP services; WS_3: it returns the medications list taken by a given patient. WS_4: it returns patients (their names) who have been administered a given medication. Now assume the following query on P1: *"Q1: what are the tests performed by patients who have been administered a medication termed as "SomeStuff"?*. Obviously the resolution of Q necessitates the composition of several services, in particular WS_1, WS_2 and WS_4 (local and remote ones).

4 MODELING ISSUES

4.1 Our Queries

Given an ontology **O(C, L, DP, OP)**, where **C** is the classes set, **L** is the literals set, **DP** is the datatype properties in O and **OP** is the object properties set in O. Our queries have the following form.
$\{ \{ ?c_1 . \Psi . p_{1.2} . ?c_2 . \Psi . p_{2.3} .... \Psi . p_{n-1.n} . ?c_n \}, CL, OS \}$, *where:*

1. $?c_1...?c_i...?c_n$ are variables of types defined by classes within *C*,
2. $p_{i.j}$ is the object property linking $?c_i$ to $?c_j$. Both $?c_{i:1\rightarrow n}$ and $p_{i.j}$ constitute the query's"**backbone**".
3. Ψ is a linking operator and it is used when one variable $?c_i$ is linked to more than one other variable such that each of these variables pose a condition on the selection of $?c_i$.
4. CL is the constraints set imposed on datatype properties of $?c_{i:1\rightarrow n}$.
5. OS is the output set, it comprises output variables (and their projected datatype properties).
In the spirit of this definition our query became:
Q1: *{?T1(Test) . [Has − Test]$^{-1}$. ?P1(Patient) . [Take-Medication] . ?M1(Medication), Ct= {$M1(Name="Some Stuff")}, Out= {?T1(Result)} }*

4.2 Data-Providing Services as Views

Web services are usually modeled with the de facto standard for service description OWL-S. In particular, OWL-S's *Service Profile* permits to model the service's functionality, inputs and outputs. However, DP Services have no explicit functionality, instead, the semantic relation holding between their inputs and outputs must be captured. Therefore OWL-S may not be the best choice for describing them since it does not allow to capture this relation. We model Data-Providing Services in our approach as *RDF Parameterized Views (PVs)* over OWL ontologies.

Each PV is a predicate $WS_i(c_i)$***:- 4-tuple <Backbone, Ct, In, Out>*** where,

$WS_i(c_i)$ is called the *view head* and it comprises the name of corresponding service and its returned results. The rest is called the *view body* and it has the following contents:
1. ***Backbone*** *it comprises both the variables set* ***C*** *(of classes types) linking the input and the output of the service, and the object properties set* ***OP*** *relating the different variables in C.*
2. ***Ct*** *is the constraints set imposed on the datatype properties of C without being required inputs of the service.*
3. ***In*** *is the necessary literals for the service invocation.*
4. ***Out*** *is the output literals.*

According to this definition, a view on one of our services is showed in figure 3, the complete list of views are presented in (Barhamgi et al., 2007).

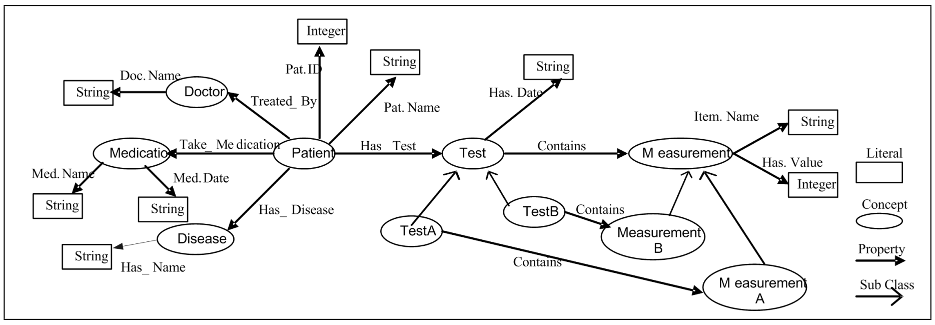


Figure 2: An example of OWL ontology modeling the peer's local data items and the items provided by its partners.

```
WS1(?TestA<Result>):-
    Backbone   = {  ?Patient . rdf:type .          Op1: Patient
                    ?Patient . Op1:has_Test . ?Test A
                    ?Test A .  rdf:type .          Op1: Test A   }
    Ct         = {     øø    }
    In         = {  ?Patient . Op1: Pat_Name . $Name      }
    Out        = {  ?Test A .  Op1:has_result .  ?A_Result  }
```

Figure 3: A Parameterized view defined for WS1.

5 WEB SERVICE-BASED QUERY REWRITING AND EXECUTION

Here we pay attention to the conditions under which a composition is considered as valid, then we present the pretreatments we apply on our views.

For formal discussion assume a query $Q(Q_{backbone}, Ct_Q, OS_Q)$ and a set of services, each has a $PV_i(Ser_{i\ backbone}, Ct_i, In_i, Out_i)$. In order to satisfy Q, backbones union of selected services has to cover the query's backbone, the final output of the composition (the sum of Out_i of selected services) should satisfy OS_Q, and the Q's constraints list Ct_Q is satisfied with the union of Ct_i. However some special cases may occur while the query rewriting process; we review them briefly.

1). The union is larger than the query backbone with provision to all of the asked outputs. Herein it should be verified whether the additional concepts in the union have a corresponding input parameter necessary for the service invocation. Here the composition cannot be invoked as a necessary input will not be available and thus the composition is invalid.

2). OS_Q is not satisfied with the sum of Out_i as some literals do not appear in the output of the composition. Herein the composition of these services is invalid.

3). A constraint specified in the Ct_Q was dropped (e.g. patient gender). Herein if dropped constraints were mandated then these services will be rejected.

4). The composition of the selected services enforce an additional constraint that was not specified in the query's Ct_Q. Herein the composition is valid.

5). There is a conflicting constraint between Q and one of the selected services (e.g. the gender property has conflicting values male vs. female). The composition herein is invalid.

6). The union of the services backbones does not cover the query backbone. In this case these services must be rejected.

All of these observations were dealt with in our Web services-based query rewriting algorithm presented in (Barhamgi et al., 2007).

5.1 Preprocessing the Defined Rdf Parameterized Views

Before the rewriting process, the parameterized views should be preprocessed. This includes the following steps.

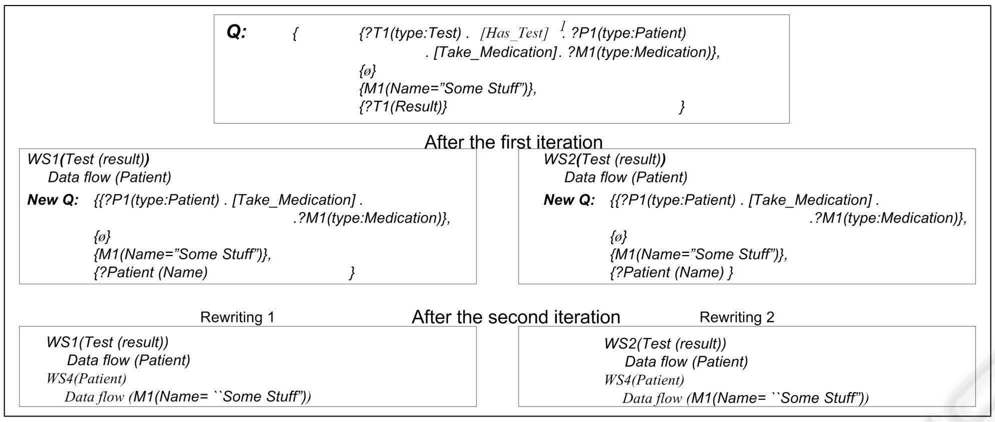
Step 1. Extending the Obtained Pvs to Reflect Owl "explicit" Subclassing Statements

We extend previously defined *PVs* with the constraints *subClassOf, subPropertyOf* that are explicitly declared in the ontology, e.g. in (figure 5, case A) a new triple was added to the *PV* of WS_1 indicating that an instance of "TestA" is also an instance of "Test".

```
WS1(?TestA<Result>):-                            WS1(?TestA<Result>):-
  Backbone = {                                     Backbone = {
        ?Patient .rdf:type . Op1: Patient                  SF1(?n) . rdf:type . Op1: Patient
        ?Patient . Op1:has_Test . SF2(?Id)                 SF1(?n) . Op1:has_Test . SF2(?Id)
        ?Test A. rdf:type . Op1: Test A                    SF2(?Id). rdf:type . Op1: Test A
        ?Test A .rdf:type . Op1: Test        }             SF2(?Id). rdf:type . Op1: Test       }
  Ct  ={øø }                                       Ct   = { øø }
  In  = {?Patient . Op1: Pat_Name . $Name  }        In   ={SF1(?n) .Op1: Pat_Name. $Name    }
  Out={?Test A .Op1:has_result. ?A_Result }        Out ={SF2(?Id) .Op1:has_result. ?A_Result}

       A. Extending PVs with                              B. Skolemizing Pvs
          subClassing constraints
```

Figure 5: An extended PV for WS1. A new triple was added (showed in bold)to reflect the relation between Test and TestA.



```
Q:      {      {?T1(type:Test) . [Has_Test] 1. ?P1(type:Patient)
                       . [Take_Medication]. ?M1(type:Medication)},
               {ø}
               {M1(Name="Some Stuff")},
               {?T1(Result)}                              }

After the first iteration

WS1(Test (result))
    Data flow (Patient)
New Q:  {{?P1(type:Patient) . [Take_Medication] .
                             .?M1(type:Medication)},
        {ø}
        {M1(Name="Some Stuff")},
        {?Patient (Name)                   }

WS2(Test (result))
    Data flow (Patient)
New Q:  {{?P1(type:Patient) . [Take_Medication] .
                             .?M1(type:Medication)},
        {ø}
        {M1(Name="Some Stuff")},
        {?Patient (Name) }

Rewriting 1        After the second iteration        Rewriting 2

WS1(Test (result))
    Data flow (Patient)
WS4(Patient)
    Data flow (M1(Name= ``Some Stuff"))

WS2(Test (result))
    Data flow (Patient)
WS4(Patient)
    Data flow (M1(Name= ``Some Stuff"))
```

Figure 4: The query rewriting process for the running example. There were two yielded rewritings for Q.

Step 2. Skolemizing Triples

Variables denoting classes in PVs need to be *skolemized* (Chen et al., 2006), that is to replace each variable by a *skolem* function helpful to merge instances stemming from different services. An example of a skolemized *PV* is shown in (figure 5, case B). The properties of a skolem function for a particular class are chosen by the domain expert.

Figure 4 shows the rewriting process of the query in the running example.

5.2 The Rewritings Execution

Our algorithm for query rewriting yields a certain number of DP services compositions. Before executing these compositions, we apply an extra algorithm in order to superimpose these compositions (if possible). The intent here is to avoid the duplicate invocation of the same service across several compositions. In our system the results of each service invocation are materialized in the form of OWL instances before being used to invoke subsequent services or sent to the requester . We need to apply extra treatment and processing over data flow among combined services. In general, three semantic operators can be applied. They are as follows.
Semantic Union ($WS_i \cup WS_j$): This operator is used to semantically combine the outputs (OWL instances) of the services WS_i, WS_j. The outcome includes the disjoint instances provided by WS_i and WS_j and the semantically equivalent instances provided by both only once.
Semantic Intersection ($WS_i \cap WS_j$): This operator can be used to return the semantically equivalent instances provided by both WS_i and WS_i.
Semantic Difference ($WS_i \oslash WS_j$): It can be used to return the instances provided by WS_i excluding the equivalent instances provided by WS_j.

Note that treatments on data flow are applied in the execution time of the composition. Returning back to our example, the execution of the compositions is done as follows. First, WS_4 is invoked with medication name. The returned patients' information is put automatically in the form of OWL instances. Then, for each obtained instance of patient we invoke both WS_1 and WS_2 and the results are materialized as OWL instances then sent to the requester.

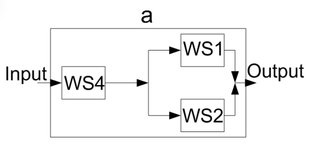


Figure 6: Data flow in Data driven Web services composition.

6 CONCLUSION

In this paper we have rapidly presented our framework (Barhamgi et al., 2007) for query resolution in terms of Data-Providing web services. We believe that this work is of great importance since that data most of the time is shared via web services in P2P collaboration environments. Currently, we are focusing taking into account the privacy constraints in our framework.

REFERENCES

Barhamgi, M., Champin, P.-A., and Benslimane, D. (2007). A Framework for Web Services-Based

Query Rewriting and Resolution in Loosely Coupled Information Systems. Technical report, http://liris.cnrs.fr/publis/?id=2647.

Chen, H., Wu, Z., Wang, H., and Mao, Y. (2006). Rdf/rdfs-based relational database integration. In *ICDE*, page 94.

Halevy, A. Y., Ives, Z. G., Madhavan, J., Mork, P., Suciu, D., and Tatarinov, I. (2004). The piazza peer data management system. *IEEE Trans. Knowl. Data Eng.*, 16(7):787–798.

Löser, A., Siberski, W., Wolpers, M., and Nejdl, W. (2003). Information integration in schema-based peer-to-peer networks. In *CAiSE*, pages 258–272.

Papazoglou, M. P. (2003). Service-oriented computing: Concepts, characteristics and directions. In *WISE*, pages 3–12.

Rodríguez-Gianolli, P., Garzetti, M., Jiang, L., Kementsietsidis, A., Kiringa, I., Masud, M., Miller, R. J., and Mylopoulos, J. (2005). Data sharing in the hyperion peer database system. In *VLDB*, pages 1291–1294.

Spyropoulou, E. and Dalamagas, T. (2006). Sdqnet: Semantic distributed querying in loosely coupled data sources. In *ADBIS*, pages 55–70.